

Cray XE6 (Raptor)

User Guide

Air Force Research Laboratory (AFRL)

DoD Supercomputing Resource Center (DSRC)

Release Notes

16 February 2011

- Initial Release

18 February 2011

- Updated cc path information
- Updated font size and color

Contents

Release Notes	2
1 Introduction.....	5
1.1 Document Scope & Assumptions	5
1.2 Policies to Review.....	5
1.3 Obtaining Accounts.....	5
1.4 Requesting Assistance	5
2 System Configuration	6
2.1 Processors.....	6
2.2 Memory.....	7
2.3 Operating System	7
2.4 File Systems	7
2.5 Peak Performance	7
3 Accessing the System	8
3.1 Kerberos	8
3.2 Logging In	8
3.3 File Transfers.....	8
4 User Environment	8
4.1 User Directories	8
4.1.1 Home Directory	8
4.1.2 Work Directory	9
4.2 Shells.....	9
4.3 Environment Variables.....	9
4.3.1 Login Environment Variables	9
4.3.2 Batch-Only Environment Variables.....	10
4.4 Modules	10
4.5 Archive Usage	11
4.5.1 Archival Command Synopsis	11
5 Program Development.....	12
5.1 Programming Models.....	12
5.1.1 Message Passing Interface (MPI)	12
5.1.2 SHared MEMory (SHMEM)	13
5.1.3 <i>Open Multi-Processing (OpenMP)</i>	15
5.1.4 <i>Hybrid MPI/OpenMP</i>	16
5.2 Available Compilers	16
5.2.1 Cray Compiler Programming Environment.....	16
5.2.2 PGI Compiler Programming Environment	17
5.2.3 GCC Compiler Programming Environment	17
5.2.4 Intel Compiler Programming Environment.....	17

5.3	Relevant Modules	17
5.4	Libraries	17
5.4.1	AMD Core Math Library (ACML)	17
5.4.2	Cray LibSci	18
5.4.3	Additional Math Libraries	18
5.5	Debuggers	18
5.5.1	gdb	18
5.5.2	TotalView	19
5.6	Code Profiling and Optimization	20
5.6.1	CrayPat	20
5.6.2	Additional Profiling Tools	21
5.6.3	Program Development Reminders	21
5.6.4	Performance Optimization Methods	21
6	Batch Scheduling	23
6.1	Scheduler	23
6.2	Queue Information	23
6.3	Interactive Logins	24
6.4	Interactive Batch Sessions	24
6.5	Batch Request Submission	24
6.6	Batch Resource Directives	25
6.7	Launch Command(s)	25
6.8	Sample Scripts	25
6.9	PBS Commands	27
6.10	Advance Reservations	27
7	Software Resources	27
7.1	Application Software	27
7.2	Useful Utilities	27
7.3	Sample Code Repository	28
8	Links to Vendor Documentation	28
8.1	Cray Links	28
8.2	SUSE Links	29
8.3	GNU, Pathscale, and Portland Group Links	29

1 Introduction

1.1 *Document Scope & Assumptions*

This document provides an overview and introduction to the use of the Cray XE6 (Raptor) located at the AFRL DSRC and a description of the specific computing environment on Raptor. The intent of this guide is to provide information to enable the average user to perform computational tasks on the system. To receive the most benefit from the information provided here, you should be proficient in the following areas:

- Use of the LINUX operating system
- Use of an editor (e.g. vi or emacs)
- Remote usage of computer systems via network or modem access
- A selected programming language and its related tools and libraries

1.2 *Policies to Review*

All policies are discussed on the [AFRL webpage](#). All users running at the AFRL DSRC are expected to know, understand, and follow the policies discussed. If you have any questions about AFRL DSRC's policies, please contact the CCAC.

1.3 *Obtaining Accounts*

Authorized DOD and Contractor personnel may request an account on Raptor through their Service Agency Approval Authority (S/AAA) and Principle Investigator (PI). Your S/AAA and PI will assign you an HPMCP project that allows you to have allocations on the HPC systems at the DSRC. As you go through the process, you may want to refer to our [step-by-step guide](#). More information can be found through the HPCMP [Portal to Information Environment \(pIE\)](#) or through the [Consolidated Customer Assistance Center \(CCAC\)](#).

1.4 *Requesting Assistance*

The Consolidated Customer Assistance Center (CCAC) is available to help users with any problems, questions, or training requirements for our HPC systems. Analysts are on duty Monday - Friday, 8:00 a.m. to 11:00 p.m. Eastern Time.

Web: http://www.ccac.hpc.mil/cust_serv_form.php

Email: help@ccac.hpc.mil

Phone: 1-877-CCAC-039 (1-877-222-2039) or 937-255-0679

Fax: 937-656-9538

2 System Configuration

Raptor is a Cray XE6. The login nodes are populated with 2.7-GHz AMD Opteron 64-bit quad-core processors. The compute nodes are populated with 2.4-GHz AMD Opteron 64-bit 8-core processors. Raptor uses a dedicated Cray Gemini communications network for MPI messages and IO traffic. Raptor uses Lustre to manage its high-speed, parallel, Infiniband RAID file system. Raptor has 2732 compute nodes that share memory only on the node; memory is not shared across the nodes. Each compute node has two 8-core processors (16 cores) with its own Compute Node Linux (CNL) operating system and 32 GBytes of DDR3 memory, with no user-accessible swap space. CNL provides many of the operating system functions available through the service nodes, although some functionality has been removed to improve performance and reduce memory usage by the system. Raptor has 344 TBytes (formatted) of disk storage.

Raptor is intended to be used as a batch-scheduled HPC system. Its login nodes are not to be used for large computational (e.g. memory, IO, long executions) work. All executions that require large amounts of system resources must be sent to the compute nodes by batch job submission.

Node Configuration		
	Login Nodes	Compute Nodes
Total Nodes	7	2732
Operating System	SUSE Linux	CNL
Cores/Node	16	16
Core Type	AMD Opteron 64-bit	AMD Opteron 64-bit
Core Speed	2.7 GHz	2.4 GHz
Memory/Node	128 GBytes	32 GBytes
User Accessible Memory/Node	126 GBytes	30 GBytes
Memory Model	Shared in node	Shared in node Distributed across cluster
Interconnect Type	Ethernet	Cray Gemini
File Systems		
File System	File System Type	Formatted Capacity
/work (\$WORKDIR file system)	Lustre	1.4 TBytes
/home (\$HOME file system)	Lustre	1.6 PBytes

Figure 1: Configuration and File System Types

2.1 Processors

Raptor uses AMD Opteron 64-bit processors on its login and compute nodes. Login nodes run at 2.7 GHz and have four quad-core processors for a total of 16 cores per node. These processors have 64 KBytes of L1 instruction cache, 64 KBytes of L1 data cache, 512 KBytes of L2 cache, and 6 MBytes of L3 cache.

Compute nodes run at 2.4 GHz and have two 8-core processors for a total of 16 cores per node. These processors have 64KBytes of L1 instruction cache, 64KBytes of L1 data cache, 512 KBytes of L2 cache, and 12 MBytes of L3 cache.

2.2 *Memory*

Raptor uses both shared and distributed memory models. Memory is shared among all the cores on a node, but is not shared among the nodes across the cluster.

Each login node contains 128 GBytes of main memory. All memory and cores on the node are shared among all users who are logged in. Therefore, users should not use more than 8 GBytes of memory at any one time.

Each compute node contains 30 GBytes of user accessible shared memory.

2.3 *Operating System*

The operating system on Raptor's login nodes is SUSE Linux 11. The compute nodes use Compute Node Linux (CNL). The combination of these two operating systems is known as the Cray Linux Environment (CLE).

2.4 *File Systems*

Raptor has the following file systems available for user storage:

/home

/home is a locally mounted Lustre file system. It has a formatted capacity of 1.6 PBytes. All users have a home directory located on this file system which can be referenced by the environment variable `$HOME`.

/workspace


/work is a locally mounted Lustre file system that is tuned for high-speed I/O performance. It has a formatted capacity of 1.4 TBytes. All users have a work directory located on this file system which can be referenced by the environment variable `$WORKDIR`.

2.5 *Peak Performance*

Raptor is rated at 34.379 HABUs and 410.04 peak TFLOPS.

3 Accessing the System

3.1 *Kerberos*

A Kerberos client kit must be installed on your desktop to enable you to get a Kerberos ticket. Kerberos is a network authentication tool that provides secure communication by using secret cryptographic keys. Only users with a valid HPCMP Kerberos authentication can gain access to Raptor. More information about installing Kerberos clients on your desktop can be found at the [CCAC Support page](#) .

3.2 *Logging In*

The preferred login to Raptor is ssh:
% ssh raptor-l0#.afrl.hpc.mil (# =1-8)

Kerberized telnet and rlogin are also allowed.

3.3 *File Transfers*

File transfers to DSRC systems must be performed using Kerberized versions of the following tools: scp, ftp, sftp, and mp scp, except file transfers to the local archive system.

4 User Environment

4.1 *User Directories*

4.1.1 Home Directory

Each user is allocated a home directory (the current working directory immediately after login) with an initial disk quota of 1 GBytes of permanent storage that is not backed up. Your home directory can be referenced locally with the `$HOME` environment variable from all nodes in the system.

You may submit a request to increase your disk space quota by contacting our Service Center. You must supply the following information for evaluation of the request by the system administrators and the AFRL DSRC management:

- Amount of system resource requested
- Length of time requested for the increase
- Special deadlines for the project
- Explanation of the attempts to work within limits

4.1.2 Work Directory


Raptor has one large file system (`/workspace`) for the temporary storage of data files needed for executing programs. You may access your personal working directory under `/workspace` by using the `$WORKDIR` environment variable, which is set for you upon login. Your `$WORKDIR` directory has no disk quotas, and files stored there do not affect your permanent file quota usage. Because of high usage, the `/workspace` file system tends to fill up frequently. Please review the [Purge Policy](#) and be mindful of your disk usage.

REMEMBER: `/workspace` is a "scratch" file system and is not backed up. You are responsible for managing files in your `$WORKDIR` by backing up files to the MSAS and deleting unneeded files when your jobs end.

All of your jobs should execute from your `$WORKDIR` directory, not `$HOME`. While not technically forbidden, jobs that are run from `$HOME` are subject to disk space quotas and have a much greater chance of failing if problems occur with that resource. Jobs that are run entirely from your `$WORKDIR` directory are more likely to complete, even if all other resources are temporarily unavailable.

If you use `$WORKDIR` in your batch scripts, you must be careful to avoid having one job accidentally contaminate the files of another job. One way to avoid this is to use the `$JOBDIR` (or `$WORK_DIR`) directory which is unique to each job on the system. The `$JOBDIR` directory is not subject to the Purge Policy until the job exits the workload management system.

4.2 Shells

The following shells are available on Raptor: `csh`, `bash`, `ksh`, `tcsh`, and `sh`. To change your default shell, use "`setenv User_Shell shell`" in your `$HOME/.personal.login` file to change your .

4.3 Environment Variables

A number of environment variables are provided by default on all HPCMP high performance computing (HPC) systems. We encourage you to use these variables in your scripts where possible. Doing so will help to simplify your scripts and reduce portability issues if you ever need to run those scripts on other systems.

4.3.1 Login Environment Variables

The following environment variables are common to both the login and batch environments:

Common Environment Variables	
Option	Purpose
<code>\$ARCHIVE_HOME</code>	Your directory on the archival system
<code>\$ARCHIVE_HOST</code>	The host name of the archival system

\$CSI_HOME	The path to the directory for the following list of heavily used application packages: ABAQUS, Accelrys, ANSYS, CFD++, Cobalt, EnSight, Fluent, GASP, Gaussian, LS-DYNA, MATLAB, and TotalView, formerly known as the Consolidated Software Initiative (CSI) list. Other application software may also be installed here by our staff.
\$DAAC_HOME	The path to the directory containing the ezViz visualization software
\$HOME	Your home directory on the system
\$JAVA_HOME	The path to the directory containing the default installation of JAVA
\$PET_HOME	The path to the directory containing the tools installed by the PET CE staff. The supported software includes a variety of open source math libraries (see BC policy FY06-01) and open source performance and profiling tools (see BC policy FY07-02).
\$SAMPLES_HOME	The path to the Sample Code Repository . This is a collection of sample scripts and codes is provided and maintained by our staff to help users learn to write their own scripts. There are a number of ready-to-use scripts for a variety of applications.
\$WORKDIR	Your work directory on the local temporary file system (i.e., local high speed disk).

4.3.2 Batch-Only Environment Variables

In addition to the variables listed above, the following variables are automatically set only in your batch environment. That is, your batch scripts will be able to see them when they run. These variables are supplied for your convenience and are intended for use inside your batch scripts.

Batch-Only Environment Variables	
Option	Purpose
\$BC_CORES_PER_NODE	The number of cores per node for the compute node on which a job is running.
\$BC_MEM_PER_NODE	The approximate maximum user-accessible memory per node (in integer MBytes) for the compute node on which a job is running.
\$BC_MPI_TASKS_ALLOC	The number of MPI tasks allocated for a job.
\$BC_NODE_ALLOC	The number of nodes allocated for a job.

4.4 Modules

Software modules are a very convenient way to set needed environment variables and include necessary directories in your path so that commands for particular applications can be found. Raptor uses "modules" to initialize your environment with system commands and libraries and compiler suites,

A number of modules are loaded automatically as soon as you log in. To see the modules which are currently loaded, run "module list". To see the entire list of available modules, run "module avail". You can modify the configuration of your environment by loading and unloading modules. For complete information on how to do this, see the [Modules User Guide](#).

4.5 *Archive Usage*

All of our HPC systems share an on-line Mass Storage Archival system (MSAS) that currently has more than 5 TBytes of Tier 1 archival storage (disk cache) and 2.2 PBytes of Tier 2 high speed archival storage utilizing a robotic tape library. The MSAS should only be used for long term storage. Every user is given an account on the MSAS.

Kerberized login and ftp are allowed into the MSAS system. Locally developed utilities may be used to transfer files to and from the MSAS as well as to create and delete directories, rename files, and list directory contents. For convenience, the environment variable `$ARCHIVE_HOME` can be used to reference your MSAS archive directory when using archive commands. The command `getarchome` can be used to display the value of `$ARCHIVE_HOME` for any user.

4.5.1 Archival Command Synopsis

A synopsis of the main archival utilities is listed below. For information on additional capabilities, see the [Archive User's Guide](#) or read the on-line man pages that are available on each system. These commands are non-Kerberized and can be used in batch submission scripts if desired.

Copy one or more files from the MSAS:

```
archive get [-C path] [-s] file1 [file2...]
```

List files and directory contents on the MSAS:

```
archive ls [lsopts] [file/dir ...]
```

Create directories on the MSAS:

```
archive mkdir [-C path] [-m mode] [-p] [-s] dir1 [dir2 ...]
```

Copy one or more files to the MSAS:

```
archive put [-C path] [-D] [-s] file1 [file ...]
```

5 Program Development

5.1 Programming Models

Raptor supports three base programming models: Message Passing Interface (MPI), SHared-MEMory (SHMEM), and Open Multi-Processing (OpenMP). A Hybrid MPI/OpenMP programming model is also supported. MPI and SHMEM are examples of the message- or data-passing models, while OpenMP only uses shared memory on a node by spawning threads.

5.1.1 Message Passing Interface (MPI)

This release of MPI-2 derives from Argonne National Laboratory MPICH-2 and implements the MPI-2.2 standard, except for spawn support, as documented by the MPI Forum in *MPI: A Message Passing Interface Standard, Version 2.2*.

On Cray systems, MPI is a component of the Cray Message Passing Toolkit (MPT), which is a software package that supports parallel programming across a network of computer systems through a technique known as message passing. MPI establishes a practical, portable, efficient, and flexible standard for message passing that makes use of the most attractive features of a number of existing message-passing systems, rather than selecting one of them and adopting it as the standard. See "man intro_mpi" for additional information.

When creating an MPI program on Raptor, ensure that the following actions are taken:

- Make sure the Message Passing Toolkit (module xt-mpt) is loaded.
To check this, run the "module list" command. If xt-mpt is not listed, use the command, "module load xt-mpt".
- Make sure the source code includes one of the following lines :

```
INCLUDE "mpif.h"    //for Fortran, or
#include <mpi.h>     //for C
```

To compile an MPI program, use the following examples:

```
ftn -o mpi_program mpi_program.f      //for Fortran,  
or  
cc -o mpi_program mpi_program.c      //for C
```

To run an MPI program within a batch script, use the following command:

```
aprun -n N mpi_program [user_arguments]
```

where *N* is the number of processes being started, with each process utilizing one core. The `aprun` command launches executables across a set of CNL compute nodes. When each member of the parallel application has exited, `aprun` exits.

A common concern for MPI users is the need for more memory for each process. By default, one MPI process is started on each core of a node. This means that on Raptor, the available memory on the node is split sixteen ways. To allow an individual process to use more of the node's memory, you need to start fewer processes. To accomplish this, do the following:

```
aprun -n N1 -n N2 mpi_program [user_arguments]
```

Here, *N1* is used as in the previous example. *N2*, however, is the number of MPI process to be started on each node. To give all of a node's memory to one process, set *N2* equal to 1.

For more information about `aprun`, see the `aprun` man page.

5.1.2 SHared MEMory (SHMEM)

These logically shared, distributed-memory access routines provide high-performance, high-bandwidth communication for use in highly parallelized scalable programs. The SHMEM data-passing library routines are similar to the MPI library routines: they pass data between cooperating parallel processes. The SHMEM data-passing routines can be used in programs that perform computations in separate address spaces and that explicitly pass data to and from different processes in the program.

The SHMEM routines minimize the overhead associated with data-passing requests, maximize bandwidth, and minimize data latency. Data latency is the length of time between a process initiating a transfer of data and that data becoming available for use at its destination.

SHMEM routines support remote data transfer through put operations that transfer data to a different process and get operations that transfer data from a different process. Other supported operations are work-shared broadcast and reduction, barrier synchronization, and atomic memory updates. An atomic memory operation is an atomic read and update operation, such as a fetch and increment, on a remote or local data object. The value read is guaranteed to be the value of the data object just prior to the update. See "man intro_shmem" for details on the SHMEM library.

When creating a SHMEM program on Raptor, ensure that the following actions are taken:

- Make sure the Message Passing Toolkit (module xt-mpt) is loaded. To check this, run the "module list" command. If xt-mpt is not listed, use the command, "module load xt-mpt"
- The source code includes one of the following lines:

```
INCLUDE 'mpp/shmem.fh'      //for Fortran, or
#include <mpp/shmem.h>      //for C
```
- The compile command includes an option to reference the SHMEM library.

To compile a SHMEM program, use the following examples:

```
ftn -lsma -o shmem_program shmem_program.f90      //for Fortran
or
Cc -lsma -o shmem_program shmem_program.c          //for C
```

Before running a SHMEM program, you may want to set the following environment variables:

```
setenv XT_LINUX_SHMEM_STACK_SIZE 24m
setenv XT_LINUX_SHMEM_HEAP_SIZE 120m
setenv XT_SYMMETRIC_HEAP_SIZE 20m
```

The program can then be launched using the aprun command as follows:

```
aprun -n N shmem_program [user_arguments]
```

where *N* is the number of processes being started, with each process utilizing one core. The aprun command launches executables across a set of CNL compute nodes. When each member of the parallel application has exited, aprun exits. For more information about aprun, see the aprun man page.

5.1.3 Open Multi-Processing (OpenMP)

OpenMP is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++ and Fortran. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. OpenMP is a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications.

When creating an OpenMP program on Raptor, ensure that the following actions are taken:

- Ensure the Message Passing Toolkit (module xt-mpt) is loaded. To check this, run the "module list" command. If xt-mpt is not listed, run "module load xt-mpt"
- If using OpenMP functions (for example, omp_get_wtime), ensure that the source code includes the line,

```
USE omp_lib
```

Or, includes one of the following:

```
INCLUDE 'omp.h'      //for Fortran, or  
#include <omp.h>     //for C
```

- The compile command includes an option to reference the OpenMP library. The PGI, PathScale, and GNU compilers support OpenMP, and each one uses a different option.

To compile an OpenMP program, use the following examples:

For C codes:

```
cc -o OpenMP_program -mp=nonuma OpenMP_program.c //PGI  
cc -o OpenMP_program -mp OpenMP_program.c        //PathScale  
cc -o OpenMP_program -fopenmp OpenMP_program.c   //GNU
```

For Fortran codes:

```
ftn -o OpenMP_program -mp=nonuma OpenMP_program.f //PGI  
ftn -o OpenMP_program -mp OpenMP_program.f        //PathScale  
ftn -o OpenMP_program -fopenmp OpenMP_program.f   //GNU
```

To run an OpenMP program within a batch script, you also need to set the \$OMP_NUM_THREADS environment variable to the number of threads in the team. For example:

```
setenv OMP_NUM_THREADS 4
aprun -n 1 OpenMP_program [user_arguments]
```

In the example above, the application starts OpenMP_program on one node and spawns a total of sixteen threads. Since Raptor has sixteen cores per compute node, this yields 1 thread per core.

5.1.4 Hybrid MPI/OpenMP

An application built with the hybrid model of parallel programming can run on Raptor using both OpenMP and Message Passing Interface (MPI).

When creating a hybrid (MPI/OpenMP) program on Raptor, follow the instructions in the MPI and OpenMP sections above for creating your program. Then use the compilation instructions for OpenMP.

To run a hybrid program within a batch script, set `$OMP_NUM_THREADS` equal to the number of threads in the team. Then launch your program using `aprun` as follows:

```
setenv OMP_NUM_THREADS 16
aprun -n N1 -d N2 hybrid_program [user_arguments]
```

Where *N1* is the number of MPI tasks and *N2* is the number of threads each task will use.

5.2 Available Compilers

The following compiler suites are available on Raptor:

- Cray compiler programming environment
- PGI compiler programming environment
- GCC compiler programming environment
- Intel compiler programming environment

5.2.1 Cray Compiler Programming Environment

This is the recommended compiler as some applications built with the other compilers have created serious system issues.

The Cray Compiler Programming Environment can be accessed by loading the module `PrgEnv-cray`.

```
Module load PrgEnv-cray
```


5.2.2 PGI Compiler Programming Environment

The PGI Compiler Programming Environment can be accessed by loading the module PrgEnv-cray.

```
Module load PrgEnv-pgi
```

5.2.3 GCC Compiler Programming Environment

The GCC Compiler Programming Environment can be accessed by loading the module PrgEnv-cray.

```
Module load PrgEnv-gcc
```

5.2.4 Intel Compiler Programming Environment

The Intel Compiler Programming Environment can be accessed by loading the module.

```
Module load PrgEnv-intel/3.0.20
```

5.3 *Relevant Modules*

This is not intended to be a comprehensive discussion of all available modules on the system, but rather a short discussion of those modules required for compiling, and how to load them. The specific information you provide is up to you, but this might include, for instance, how modules are used at your center, information to help users choose between specific modules, or an explanation of the naming convention used for different versions of a module. If certain modules are “preferred” or “default” it might also be helpful to note them here.

For more information on using modules, see the [Modules User Guide](#).

5.4 *Libraries*

5.4.1 AMD Core Math Library (ACML)

Raptor provides the AMD Core Math Library (ACML). ACML is a set of numerical routines tuned specifically for AMD64 platform processors. The routines, which are available via both FORTRAN and C interfaces, include:

- Basic Linear Algebra Subroutines (BLAS) - Levels 1, 2, and 3
- Linear Algebra Package (LAPACK)
- Fast Fourier Transform (FFT) routines for single-precision, double-precision, single-precision complex, and double-precision complex data types

- Random Number Generator
- Fast Math and Fast Vector Library

The routines in the ACML can be accessed by including the library reference on your compile command line. For example, `ftn -l acml fort.f90`

5.4.2 Cray LibSci

In addition to ACML, Raptor provides Cray's LibSci library as part of the Cray Programming Environment. This library is a collection of single-processor and parallel numerical routines that have been tuned for optimal performance on Cray XT systems. The LibSci library is loaded by default and contains optimized versions of many of the BLAS math routines as well as Cray versions of most of the ACML routines. Users should call the LibSci versions, instead of the public domain or user written versions, to optimize application performance on Raptor.

The routines in LibSci are automatically included when using the `ftn`, `cc`, or `CC` commands. You do not need to use the `"-l sci"` flag in your compile command line.

Cray LibSci includes the following:

- Basic Linear Algebra Subroutines (BLAS) - Levels 1, 2, and 3
- Linear Algebra Package (LAPACK)
- Scalable LAPACK (ScaLAPACK) (distributed-memory parallel set of LAPACK routines)
- Basic Linear Algebra Communication Subprograms (BLACS)
- Iterative Refinement Toolkit (IRT)
- SuperLU (for large, sparse nonsymmetrical systems of linear equations)

5.4.3 Additional Math Libraries

There is also an extensive set of Math libraries available in the `$PET_HOME/MATH` directory on Raptor. Information about these libraries may be found on the Baseline Configuration Web site at [BC policy FY06-01](#).

5.5 *Debuggers*

5.5.1 `gdb`

The GNU Project Debugger (`gdb`) is a source level debugger that can be invoked either with a program for execution or a running process id. To launch your program under `gdb` for debugging, use:

```
gdb a.out corefile
```

To attach gdb to a program that is already executing on this node, use the following command:

```
gdb a.out pid
```

For more information, the GDB manual can be found at <http://www.gnu.org/software/gdb/> .

5.5.2 TotalView

TotalView is a debugger that supports threads, MPI, OpenMP, C/C++, and Fortran, mixed-language codes, advanced features like on-demand memory leak detection, other heap allocation debugging features, and the Standard Template Library Viewer (STLView). Unique features like dive, a wide variety of breakpoints, the Message Queue Graph/Visualizer, powerful data analysis, and control at the thread level are also available.

Follow these steps to use TotalView on Raptor via a UNIX X-Windows interface:

1. Ensure that an X server is running on your local system. Linux users will likely have this by default, but MS Windows users will need to install a third party X Windows solution. There are various options available.
2. For Linux users, connect to Raptor using `ssh -Y`. Windows users will need to use PuTTY with X11 forwarding enabled (Connection→SSH→X11→Enable X11 forwarding).
3. Compile your program on Raptor with the "-g" option.
4. Submit an interactive job:

```
qsub -l ncpus=4 -A project_ID -l walltime=00:30:00 -q  
debug -v DISPLAY -l
```

5. After a short while the following message will appear:
qsub: waiting for job NNNNNNNN to start
qsub: job NNNNNNNN ready
6. You are now logged into an interactive batch session.
7. Load the TotalView module:

```
module load totalview
```

8. Start program execution:

```
Totalview apron -a-n 4 ./my_mpi_prog.exe arg1 arg2...
```

9. After a short delay, the TotalView windows will pop up. Click "GO" to start program execution.

For more information on using TotalView, see the [TotalView Documentation](#) page.

5.6 *Code Profiling and Optimization*

Profiling is the process of analyzing the execution flow and characteristics of your program to identify sections of code that are likely candidates for optimization, which increases the performance of a program by modifying certain aspects for increased efficiency.

We provide CrayPat to assist you in the profiling process. We have also included (below) a basic overview of optimization methods with information about how they may improve the performance of your code. More information on optimization can be found in [Performance Optimization Methods](#) (below).

5.6.1 CrayPat

CrayPat is an optional performance analysis tool used to evaluate program behavior on Cray supercomputer systems. CrayPat consists of the following major components: pat_build, pat_report, and pat_help. The data produced by CrayPat can also be used with Cray Apprentice2, an analysis tool that is used to visualize and explore the performance data captured during program execution. Man pages are available for pat_build, pat_report, pat_help, and Apprentice2. Additional information can be found in the document, "[Using Cray Performance Analysis Tools](#)".

The following steps should get you started using CrayPat:

1. Load the "xt-craypat" module

```
module load xt-craypat
```

2. Recompile the code as you normally would to generate an executable.

```
ftn mycode.f90 -o mycode
```

3. Use the `pat_build` command to generate an instrumented executable.

```
pat_build -g mpi -u mycode
```

This generates an instrumented executable called `mycode+pat`. Here the "-g" option enables the "mpi" tracegroup. See "man `pat_build`" for available tracegroups.

4. Run the instrumented executable with `aprun` via PBS.

```
aprun -n 4 ./mycode+pat
```

This generates an instrumented output file (e.g. `mycode+pat+2007-12tdt.xf`).

5. Use `pat_report` to display the statistics from the output file

```
pat_report mycode+pat+2007-12tdt.xf > mycode.pat_report
```

Additional profiling options are available. See "man `pat_build`" for additional instrumentation options.

5.6.2 Additional Profiling Tools

There is also a set of profiling tools available in the `$PET_HOME/pkgs` directory on Diamond. Information about these tools may be found on the Baseline Configuration Web site at [BC policy FY07-02](#).

5.6.3 Program Development Reminders

If an application is not programmed for distributed memory, then only the cores on a single node can be used. This is limited to 4 cores on Raptor. Keep the system architecture in mind during code development. For instance, if your program requires more memory than is available on a single node, then you will need to parallelize your code so that it can function across multiple nodes.

5.6.4 Performance Optimization Methods

Optimization generally increases compilation time and executable size, and may make debugging difficult. However, it usually produces code that runs significantly faster. The optimizations that you can use will vary depending on your code and the system on which you are running.

Note: Before considering optimization, you should always ensure that your code runs correctly and produces valid output.

In general, there are four main categories of optimization:

- Global Optimization

- Loop Optimization
- Inter-Procedural Analysis and Optimization(IPA)
- Function Inlining

5.6.4.1 Global Optimization

A technique that looks at the program as a whole and may perform any of the following actions:

- Performed on code over all its basic blocks
- Performs control-flow and data-flow analysis for an entire program
- Detects all loops, including those formed by IF and GOTOs statements and performs general optimization.
- Constant propagation
- Copy propagation
- Dead store elimination
- Global register allocation
- Invariant code motion
- Induction variable elimination

5.6.4.2 Loop Optimization

A technique that focuses on loops (for, while, etc.,) in your code and looks for ways to reduce loop iterations or parallelize the loop operations. The following types of actions may be performed:

- Vectorization – rewrites loops to improve memory access performance. Some compilers may also support automatic loop vectorization by converting loops to utilize low-level hardware instructions and registers if they meet certain criteria.
- Loop unrolling – (also know as "unwinding") replicates the body of loops to reduce loop branching overhead and provide better opportunities for local optimization.
- Parallelization – divides loop operations over multiple processors where possible.

5.6.4.3 Inter-Procedural Analysis and Optimization (IPA)

A technique that allows the use of information across function call boundaries to perform optimizations that would otherwise be unavailable.

5.6.4.4 Function Inlining

Function Inlining is a technique that seeks to reduce function call and return overhead. It is:

- Used with functions that are called numerous times from relatively few locations.
- Allows a function call to be replaced by a copy of the body of that function.
- May create opportunities for other types of optimization
- May not be beneficial.

Improper use of this form of optimization may increase code size and actually result in less efficient code.

6 Batch Scheduling


6.1 Scheduler

The Portable Batch System (PBS) is currently running on Raptor. It schedules jobs and manages resources and job queues, and can be accessed through the interactive batch environment or by submitting a batch request. PBS is able to manage both single-processor and multiprocessor jobs.

6.2 Queue Information

The following table describes the PBS queues available on Raptor:

Priority	Queue Name	Job Class	Max Wall Clock Time	Max Cores Per Job	Comments
Highest	test	N/A	48 Hours	21856	Staff only testing
	urgent	Urgent	168 Hours	128	Jobs belonging to DoD HPCMP Urgent Projects.
	debug	Debug	1 Hour	21856	User testing
	high	High	168 Hours	21856	Jobs belonging to DoD HPCMP High Priority Projects.
	challenge	Challenge	168 Hours	21856	Jobs belonging to DoD HPCMP Challenge Projects.



	standard	Standard	168 Hours	21856	Standard jobs
	transfer	N/A	12 Hours	16	Data transfer for user jobs
Lowest	background	Background	120 Hours	32	Unrestricted Access – no allocation charge

Figure 9: Queue Information

6.3 *Interactive Logins*

When you log in to Raptor, you will be running in an interactive shell on a login node. The login nodes provide login access for Raptor and support such activities as compiling, editing, and general interactive use by all users. Please note the [Login Node Abuse](#) policy. The preferred method to run resource intensive executions is to use an interactive batch session.

6.4 *Interactive Batch Sessions*

In order to use the interactive batch environment, you must first acquire an interactive batch shell. This is done by executing a qsub command with the "-I" option from within the interactive environment. For example,

```
qsub -l ncpus=# -A project_ID -q queue_name -l walltime=wall_time -I
```

Your batch shell request will be placed in the desired queue and scheduled for execution. This may take a few minutes because of the system load. Once your shell starts, you can run or debug interactive applications, execute job scripts, start an execution on the compute nodes via the aprun command, or postprocess data, etc.

6.5 *Batch Request Submission*

PBS batch jobs are submitted via the qsub command. The format of this command is:

```
qsub [ options ] batch_script_file
```

qsub options may be specified on the command line or imbedded in the batch script file by lines beginning with "#PBS".

For a more thorough discussion of PBS Batch Submission, see the [PBS User Guide](#).

6.6 Batch Resource Directives

Batch resource directives allow you to specify to PBS how your batch jobs should be run, and what resources your job requires. Although PBS has many directives, you only need to know a few to run most jobs.

The basic syntax of PBS directives is as follows:

```
#PBS option[ [=]value]
```

where some options may require values to be included. For example, to set the number of cores for your job, you might specify the following:

```
#PBS -l ncpus=8
```

The following directives are required for all jobs:

Option	Value	Description
-A	project_ID	Name of the project
-q	queue_name	Name of the queue
-l	ncpus=#	Number of cores
-l	walltime=HH:MM:SS	Maximum wall time

Figure 10: Required Directives

A more complete listing of batch Resource Directives is available in the [PBS User Guide](#).

6.7 Launch Command(s)

This should describe the command(s) that may be used on this system to launch executables onto the compute nodes (i.e., mpiexec_mpt, aprun, etc.). It should also show all required options for those commands or link to examples.

6.8 Sample Scripts

While it is possible to include all PBS directives at the qsub command-line, the preferred method is to embed the PBS directives within the batch request script using "#PBS". The following is a sample batch script:

```

# #!/bin/bash

# Declare the project under which this job run will be charged. (required)
# Users can find eligible projects by typing "show_usage" on the command line.
#PBS -A project_ID

# Request 1 hour of wallclock time for execution (required).
#PBS -l walltime=01:00:00

# Request 4 cores (required).
#PBS -l ncpus=4

# Submit job to debug queue (required).
#PBS -q debug

# Declare a jobname.
#PBS -N myjob

# Send standard output (stdout) and error (stderr) to the same file.
#PBS -j oe

# Make a new subdirectory in working storage space.
mkdir $WORKDIR/projA-7

# Change to the new directory.
cd $WORKDIR/projA-7

# Check MSAS availability. If not available, then wait.
archive stat -s

# Retrieve executable program from the MSAS.
archive get -C $ARCHIVE_HOME/project_name program.exe

# Retrieve input data file from the MSAS.
archive get -C $ARCHIVE_HOME/project_name/input data.in

# Execute a parallel program.
aprun -n 4 my_program < data.in > projA-7.out

# Check MSAS availability. If not available, then wait.
archive stat -s

# Create a new subdirectory on the MSAS.
archive mkdir -C $ARCHIVE_HOME/project_name output7

# Transfer output file back to the MSAS.
archive put -C $ARCHIVE_HOME/project_name/output7 projA-7.out

# Clean up unneeded files from working storage.
cd $WORKDIR
rm -r projA-7

```

Additional examples are available in the [PBS User Guide](#) and in the Sample Code Repository (`$SAMPLES_HOME`) on Raptor.

6.9 *PBS Commands*

The following commands provide the basic functionality for using the PBS batch system:

qsub: Used to submit jobs for batch processing.

```
qsub [...qsub options...] my_job_script
```

qstat: Used to check the status of submitted jobs.

```
qstat PBS_JOBID //check one job
```


```
qstat -u my_user_name //check all of user's jobs
```

qdel: Used to kill queued or running jobs.

```
qdel PBS_JOBID
```

A more complete list of PBS commands is available in the [PBS User Guide](#).

6.10 *Advance Reservations*

An Advance Reservation Service (ARS) is available on Raptor for reserving up to 10,928 cores for use, starting at a specific date/time, and lasting for a specific number of hours. The ARS is accessible via most modern web browsers at <https://reservation.hpc.mil/> . Authenticated access is required. An ARS User's Guide is available online once you have logged in.

7 Software Resources

7.1 *Application Software*

All COTS software packages can be found in the `$CSI_HOME` (`/usr/local/applic`) directory. A complete listing with installed versions can be found on our [software page](#). The general rule for all COTS software packages is that the two latest versions will be maintained on our systems. For convenience, modules are also available for most COTS software packages.

7.2 *Useful Utilities*

The following utilities are available on Raptor:

Utility	Description
archive	Perform basic file-handling operations on the MSAS
blocker	Convert a file to fixed-record-length format
bull	Display the system bulletin board
cal2jul	Convert a date into the corresponding Julian day

Utility	Description
datecalc	Print an offset from today's date in various formats
extabs	Expand tab characters to spaces in a text file
getarchome	Display the value of ARCHIVE_HOME for a given userid
getarchost	Display the value of ARCHIVE_HOST for a given userid
getprojhome	Display the archival directory path for a subproject
getprojhost	Display the archival host address for a subproject
justify	Justify a character string with padding
lss	Show unprintable characters in file names
mpibzip2	Parallel implementation of the bzip2 compression utility
mpscp	High-performance remote file copy
qlim	Report current batch queue usages and limits.
qpeek	Display spooled stdout and stderr for an executing batch job.
qview	Display information about batch jobs and queues
show_queues	Report current batch queue status, usage, and limits
show_storage	Display MSAS allocation and usage by subproject
show_usage	Display CPU allocation and usage by subproject
stripdos	Strip DOS end-of-record control characters from a text file
stripes	Report the OST striping pattern of files in a Lustre filesystem
tails	Display the last five lines of one or more files
tree	Display the subdirectory tree for a selected directory name
trim	Trim trailing blanks from text file lines
vman	Browse an on-line man page using the view command
xt_free	Display the amount of free and used memory for login nodes

Figure 11: Local Utilities


7.3 *Sample Code Repository*

The Sample Code Repository is a directory that contains examples for COTS batch scripts, building and using serial and parallel programs, data management, and accessing and using serial and parallel math libraries. The `$SAMPLES_HOME` environment variable contains the path to this area, and is automatically defined in your login environment.

8 Links to Vendor Documentation

8.1 *Cray Links*

Cray Home: <http://docs.cray.com/> 

Cray Application Developer's Environment User's Guide
<http://docs.cray.com/books/S-2396-50/S-2396-50.pdf> ,

8.2 SUSE Links


Novell Home: <http://www.novell.com/linux/> 

Novell SUSE Linux Enterprise Server: <http://www.novell.com/products/server/> 

8.3 GNU, Pathscale, and Portland Group Links

GNU Home: <http://www.gnu.org> 

GNU Compiler: <http://gcc.gnu.org> 

Pathscale Compiler Documentation: <http://www.pathscale.com/node/70> 

Portland Group Resources Page: <http://www.pgroup.com/resources/> 